```
//NOTE 1 VAR LET CONST:

// var --> is accessible inside the function in which its defined, even outside of scope
//let --> is accessible inside the block in which its defined
//conts --> constant variable

// exp:
function sayHello(){
  for (var i = 0; i < 2; i++) {
    console.log(i);
  }
  console.log(i);
}
sayHello();
//result:
//0
//1
//2

//exp 2:
function sayHello(){
  for (let i = 0; i < 5; i++) {
    console.log(i);
  }
  console.log(i);
}
sayHello();
//result:
//error because i pada console.log(i) yang kedua diluar scope let


// NOTE 2 OBJECTS:

//object in js is collection of key-value pairs
//in js method is also an object

const person = {
  name : "Hammam",
  walk : function(){}, //cara lama
  walk(){} //cara ES6
};


// NOTE 3 THIS:

//this is js behave diffenrenty from other programming language
//value of "this" is determined by how a function is called

const person = {
  name : "Hammam",
  walk(){
    console.log(this);
    //look at this
  }
};

//if we call a function as a method in an object, "this" always return a reference of that object
person.walk();
//result:
//{name: "Hammam", walk: ƒ}

//but if we call function as stanalone object or outside of an object, "this" will return the global object which is the windows object on browser
const walk = person.walk;
walk();
//result:
//Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, frames: Window, …}


// NOTE 4 BINDING(MENGIKAT) 'THIS':

//guna : fixing pengglobalan 'this' jika dipanggil sendiri atau diluar object
//remember in js, function is an object
//exp:
const person = {
  name : "Hammam",
  walk(){
    console.log(this);
    //look at this
  }
};

const walk = person.walk.bind(person);
//which means: the bind method will return a new instance of walk function and set 'this' to person object.
walk();
//result:
//{name: "Hammam", walk: ƒ}
//bind method can make value of 'this' permanently


// NOTE 5 ARROW FUNCTION:

//exp1:
//cara lama:
const square = function (number){
  return number * number;
}
//cara baru1:
const square2 = (number) => {
  return number * number;
}
//cara baru2:
const square3 = (number) => number * number;
console.log(square(5));
console.log(square2(5));
console.log(square3(5));
//al result : 25

//exp2:
const kadaluarsa = [
  {id:1, isActive: true},
  {id:2, isActive: true},
  {id:3, isActive: false},
];

//cara lama
const ada = kadaluarsa.filter(function(obj){
    return obj.isActive;
})
//cara baru pake arrow function
const ada2 = kadaluarsa.filter((obj) => obj.isActive);

console.log(ada);
console.log(ada2);

//ttg filter() method:
//syntax :
array.filter(method_yang_berisi_persyaratan_anggota_array_lulus_filter);
//exp:
const ada2 = kadaluarsa.filter((obj) => obj.isActive);
//kadaluarsa = arraynya
//obj = anggota2 arraynya
//obj.isActive = hanya yg true saja yg lolos filter dan akan dimasukkan ke varaiable ada2


// NOTE 6 ARROW FUNCTION AND THIS:

exp:
const person = {
  talk(){
    setTimeout(function(){ //callback function
      console.log(this);
    },1000);
  }
};
person.talk();
result:
//Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, frames: Window, …}

//how this can happen?
```

```
//because that callback function is not part of any object so its not like the talk method in the person object, its a standalone function

//how to solve? which means how can we have a reference to the person object inside of this callback function?

//cara 1:
//declare a variable an set it to 'this' outside of the callback function
const person = {
  talk(){
    const a = this;
    setTimeout(function(){ //callback function
      console.log(a);
    },1000);
  }
};
person.talk();

//cara 2:
//using arrow function. because arrow doesnt rebind the 'this' keyword. which means arrow function will inherit that 'this' keyword in which this code is defined.
const person = {
  talk(){
    setTimeout(() => { //callback function
      console.log(this);
    },1000);
  }
};
person.talk();


// NOTE 7 ARRAY.MAP():

//guna : in react, it used to render lists.

//exp:
//cara lama:
const colors = ['red','green', 'blue'];
const item = colors.map((warna) => '<li>' + warna + '</li>');
                          //concetination cara lama
//cara baru:
//pake template literal
//a template literal is a way to concatenate strings while allowing embedded expressions and improving readability.
//With template literals, we use enclosing back-ticks ( ` ) instead
const colors = ['red','green', 'blue'];
const item = colors.map((warna) => `<li>${warna}</li>`);

console.log(item);
//result:
// 0: "<li>red</li>"
// 1: "<li>green</li>"
// 2: "<li>blue</li>"

// NOTE 8 OBJECT DESTRUCTURING:

const address = {
  street : 'slamet riyadi',
  city : 'surakarta',
  country : 'indonesia'
};

// cara lama
const street = address.street;
const city = address.city;
const country = address.country;

//cara destructuring:
const {street, city, country} = address;
//cara destructuring (jika kita pengen nama variabel beda):
const {street:jalan, city:kota, country:negara} = address;

console.log(street, city, country);
//result:
// slamet riyadi surakarta indonesia
console.log(jalan,kota,negara);
//result;
// slamet riyadi surakarta indonesia


//NOTE 9 SPREAD OPERATOR:

//guna spread: to get each individual item in that array or object

//exp 1 in array:
const first = [1,2,3];
const second = [4,5,6];

//gabungin array cara lama:
const combined = first.concat(second);
//gabungin array pakai spread:
const combined2 = [...first,...second];
//keuntungan lain: bisa nambahin di tengah2nya:
const combined3 = [...first,6,10,...second,13];

console.log(combined);
//result:
// (6) [1, 2, 3, 4, 5, 6]
console.log(combined2);
//result:
// (6) [1, 2, 3, 4, 5, 6]

//exp 2 in object:
const first = { nama: 'hammam'};
const second = {asal: 'solo'};

const combined = {...first,...second};
console.log(combined);
//result:
// {nama: "hammam", asal: "solo"}


// NOTE 10 CLASSES:
class Person {
  constructor(nama){
    this.nama = nama;
  }
  walk(){
    console.log('berjalan');
  }
}
const orang = new Person('hammam');

console.log(orang.nama);
orang.walk();


// NOTE 11 INHERITANCE:

class Person {
  //constructor standar lha
  constructor(nama){
    this.nama = nama;
  }
  walk(){
    console.log('berjalan');
  }
}

class Teacher extends Person {
  //mbikin constructor di children
  constructor(nama, degree = null){
    super(nama);
    this.degree = degree;
  }

  teach(){
    console.log("tech");
  }
}

const guru = new Teacher('hammam', 's2');

console.log(guru.nama);
console.log(guru.degree);
guru.walk();
```

```
guru.teach();                                   //look ada tambahan default disini


// NOTE 12 MODULES:
//when working with module, the object we define in a module
are private by default,so they are not accessible from the
outside
//to make that public, you must export it

//exp:
//buat 2 file berbeda: person.js and teacher.js

//1. person.js
export class Person {
  //constructor standar lha
  constructor(nama){
    this.nama = nama;
  }
  walk(){
    console.log('berjalan');
  }
}

//2. teacher.js
//import clss Person in person.js file
import {Person} from './person.js';
export class Teacher extends Person {
  //mbikin constructor di children
  constructor(nama, degree = null){
    super(nama);
    this.degree = degree;
  }

  teach(){
    console.log("tech");
  }
}

//import class Teacher
import {Teacher} from './teacher.js';
const guru = new Teacher('Hammam', 'S3')
console.log(guru.nama, guru.degree);


// NOTE 13 NAMED AND DEFAULT EXPORT:

//Named exports are useful to export several values. During the
import, one will be able to use the same name to refer to the
corresponding value.
//Concerning the default export, there is only a single default
export per module. A default export can be a function, a class,
an object or anything else. This value is to be considered as the
“main” exported value since it will be the simplest to import.

//exp Named export:
//exp 1:
// imports
// ex. importing a single named export
import { MyComponent } from "./MyComponent";
// ex. importing multiple named exports
import { MyComponent, MyComponent2 } from
"./MyComponent";
// ex. giving a named import a different name by using "as":
import { MyComponent2 as MyNewComponent } from
"./MyComponent";
// yang pengen di import
export const MyComponent = () => {}
export const MyComponent2 = () => {}

//exp 2: Import all the named exports onto an object:
import * as MainComponents from "./MyComponent";
// use MainComponents.MyComponent and
MainComponents.MyComponent2 here

//exp Default export:
// import
import MyDefaultComponent from "./MyDefaultExport";
// export
const MyComponent = () => {}
export default MyComponent;
```